Java Practice Test

Thursday 14th December 2017

10.00 – 13.00
THREE HOURS

- Please make your swipe card visible on your desk.
- After the planning time log in using your username as **both** your username and password.

The maximum total is 25.

Credit will be awarded throughout for clarity, conciseness, useful commenting, and appropriate use of **assertions**.

Important notes:

- THREE MARKS will be deducted from solutions that do not compile. You should comment out any code which you cannot get to compile.
- ONE MARK will be deducted from solutions that include any additional print statements. All output should be provided or look like the sample run. Please make sure you remove any of your additional print statements before submitting your final solution.

Problem Description

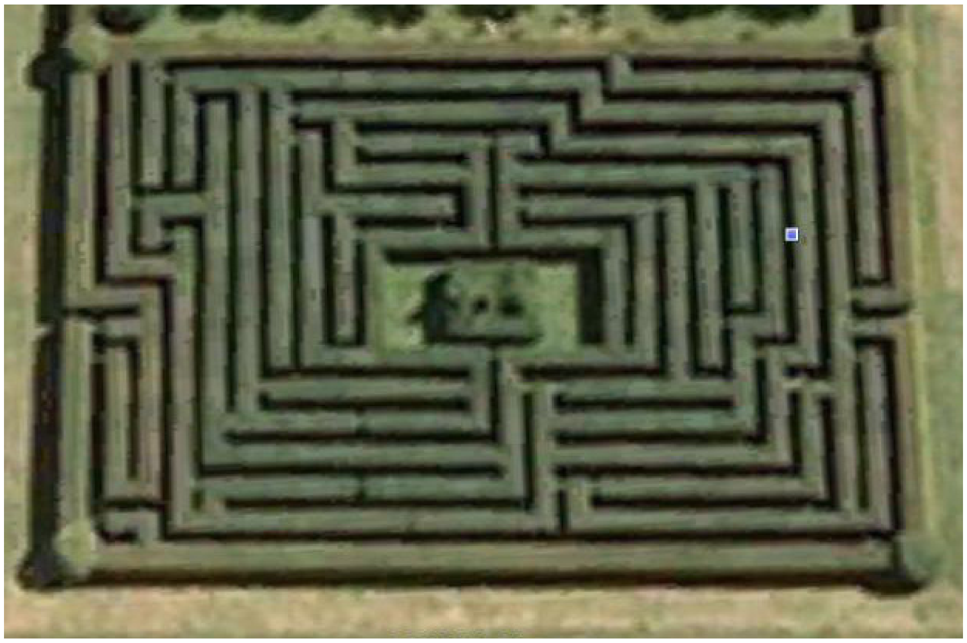

Figure 1: An aerial view of the hedge maze at Hatfield House (with thanks to Google Earth).

.

A hedge maze[1] is an outdoor maze or labyrinth in which the “walls” between passages are made of vertical hedges. Hedge mazes were a popular form of amusement for British and European royalty and aristocrats in the 17th and 18th centuries. Although less numerous (and less elitist) today, there are currently more than 100 mazes open to the public in Britain at venues such as Hampton Court Palace, Leeds Castle and Hatfield House (see Figure 1).

As shown in Figure 2, we will represent a maze as a rectangular two-dimensional array of characters, using the “barrier” characters '`-`' and '`|`' for hedges and (unique) “marker” characters to denote points of interest; here '`>`' represents the entrance, '`X`' the exit, and '`U`' an unreachable square.

A solution path through the maze is a sequence of directions from one marker character to another using the letters N, S, E and W for north, south, east

[1]A desirable accessory if you are ever lucky enough to own a luxurious castle or a stately country manor.

```
  012345678
0 ---------
1 >     |U|
2 - --- ---
3 | | |   |
4 - - -----
5 |   |   |
6 - ----- -
7 |       X
8 ---------
```

```
  012345678
0 ---------
1 ##    |U|
2 |#|-| |-|
3 |#| |   |
4 |#| ----|
5 |#  |   |
6 |#----- |
7 |########
8 ---------
```

Figure 2: A textual representation of a maze with marker characters and a valid solution path from the entrance to the exit (right).

and west respectively. The path must not pass through any hedges or pass outside the boundaries of the maze. Hence a valid solution path for the example maze from the entrance to the exit (shown on the right in Figure 2) is ESSSSSSEEEEEEE. Although it leads from the entrance to the exit, the path EEEEEEEESSSSSS is not a valid solution path since it passes through hedges.

Pre-supplied files and methods

To get you started, you are initially supplied with five files: `Coords.java`, `Dir.java`, `Maze.java`, `Solution.java`, and `Tests.java`. `Coords` is the class that you are to use for holding the pair of coordinates of a position in the maze and `Dir` is an enumerated type for directions (N,S,E,W).

Feel free to extend these two or not use all the functionality provided.

Some of the questions below ask you to add some more methods to the `Maze` class. Some will ask you to add methods to `Solution`.

The `Tests` class is provided to help you test your code. Currently its `main` method is commented out. Uncomment the code as you provide functionality that can be tested. You may find it helpful to fill in the missing tests cases and add additional test cases to `Tests.java`. We will not be using your `Tests` class to test your code so it can be left in whatever state you find useful.

Please read the files, before designing any code. You may use methods that you have written in later methods and you may write any additional methods you need to do the tasks.

Specific Tasks

1. Extend the following methods in the `Maze` class:

   (a) A method `isInMaze` that takes a pair of coordinates and returns `true` if and only if the coordinates are within the maze. For example, if your maze is an 8 by 8 square then (0,7) should return `true` and (8,1) should return `false`. *3 marks*

   (b) A method `findMarker` which finds the coordinates of marker character `ch` in the `maze`. The method should return the coordinates of the position in a `Coords`. If the marker is not in the maze, both the row and column coordinates should be set to `-1`.

   For example, using the maze of Figure 2, the call:

   ```
   c = findMarker('X');
   ```

   should result in `c.row` having the value 7, and `c.col` having the value 8. *3 marks*

   (c) A method `next` which takes as parameters coordinates and a direction and returns for the given direction the coordinates of the next position in the maze. This method is not expected to test whether the space in the maze is occupied or whether it is even inside the maze.

   For example, assuming you have a variable `c` whose value is the coordinates (1,2) and the northern direction the call:

   ```
   c1 = next(c, Dir.N);
   ```

   should result in `c1.row` having the value 0, and `c1.col` having the value 2. *3 marks*

   (d) A method `isPossible` that takes a pair of coordinates and returns true if and only if the maze at that position is a place one can move to. The start and stop places are considered to be possible, as is any space in the maze. To be possible the coordinates have to be in the maze. It is not possible to move where there is a hedge (`'-'`,`'|'`) character. *3 marks*

2. Extend the following methods in the `Solution` class:

   (a) A method `isValidPath` which determines if a given path through a maze leads from the entrance marker '`>`' to the exit marker '`X`' without moving outside the boundaries of the maze or passing through a hedge.

   For example, using the path that represents ESSSSSSEEEEEEE and the simple maze of Figure 2, `isValidPath` should return `true` whereas the SSSSSSSEEEEEEE path should return `false`. *5 marks*

   (b) A method `putPathInMaze` which replaces the start, finish, and appropriate empty spaces with `'#'` characters. *3 marks*

   (c) A method `findPath` which finds a valid solution path through a maze beginning at a marker character `start` and terminating at a marker character `end`. It is possible that there isn't a path through the maze. Your code will be tested on both mazes with dead ends and those without. Dealing with dead ends is hard and will carry 2 of the 5 marks. (So if you cannot implement backtracking, make sure your code can deal with mazes without dead ends such as `TinyMaze` and mazes with no solutions such as `TinyNoStartMaze` - both in `Tests.java`.)

   For bonus marks make sure your path does not have any unnecessary steps, that is, it should not have any sub-paths that don't make any progress such as `EW` or `ENWS` in the path. To get the bonus marks, in addition to producing minimal paths, you must fill in the comment above `findPath` explaining how you ensured that your paths were minimal. *5 +* **2 bonus** *marks*

There are several maze examples in `Tests.java` available for you to use. `goodMaze` is the example from this test paper. If you are happy with the results of your tests for each method and want to see your methods working together you could call `putItAllTogether` in `Tests.java`.

If you execute `solve` from `Solution.java` on `goodMaze` your outcome should be:

```
The maze to be traversed is:
  012345678
0 ---------
1 >     |U|
2 | |-| |-|
3 | | |   |
4 | | ----|
5 |   |   |
6 | --|-- |
7 |       X
8 ---------

A path through the maze from '>' to 'X' is:
ESSSSSSEEEEEEE

   012345678
0 ---------
1 ##    |U|
2 |#|-| |-|
3 |#| |   |
4 |#| ----|
5 |#  |   |
6 |#----- |
7 |########
8 ---------
```

Hints:

1. You will save a lot of time if you begin by carefully studying the files provided.
2. Feel free to define any auxiliary functions which would help to make your code more elegant.
3. One (but by no means the only) way to solve the final question for potentially dead ended mazes is to use a helper function which implements a "breadth first" algorithm based on your current position within the maze. First, for the base case consider:
   (a) How will you know when you have solved the maze?

   (b) How will you know when your current position is definitely not en route to a valid solution?

   Next, use recursive calls to see if going North, South, East or West from your current position leads to a valid solution. Concurrently, you will need to maintain the string of taken directions, and to mark the position of explored squares on the maze (remembering to unmark them when dead ends are encountered).

4. Try to attempt all questions. If you cannot get one of the questions to work, try the next one.

Important note: You should not change the filenames or move any files out of their original location, otherwise the autotesting will fail.